Table of contents

Introduction & Game Rules

Name / Based on: The Joker Game / Poker Cards

Module used: random, time

Players: 1 human player & (1 ~ 3) computer players

Form of I/O: Console standard input/output

Winning condition:

There is no card in player's hand.

Losing condition:

All the other players win, then the one left with 2 jokers in hand loses.

Steps of the game:

During the game, if the losing condition is satisfied, then the game ends.

1. The human player types his name and chooses how many computer players to play with.
2. Distribute 54 cards randomly among the players at first.
3. All the players throw out pairs in their hand.
4. The human player chooses a computer player to draw a card from.
5. The human player throws out pairs in his hand.
6. All the computer players draw a card from the others.
7. All the computer players throw out pairs in their hand.

Algorithms

Sorting cards:

Time of use: Before throwing out pairs

Based on: Quicksort

Details:

Before the player throws out pairs, it is necessary to sort the cards in order to keep the throwing pairs algorithm to run (it will be elaborated later in this section). The value of a card is determined by the ***card_value*** function which assigns 1. If the card is a number, then the value is the number, 2. If the card is a string (e.g. ‘Q’, ‘Joker’ etc.), then the value is the sum of ascii values of the string. Then the ***partition***, ***q_sort*** and ***my_quick_sort*** functions implement quicksort with ***card_value*** determining the value of an element.

Throwing out pairs:

Time of use: After sorting cards

Details:

Check the values of the card with index (i) and index (i + 1), if they are the same, then remove both cards and keep the index i for next two cards. Otherwise, increase the index by one (+1). It will loop through all the cards eventually, and there will be no pairs among the remaining ones.

Algorithms

Drawing card algorithm of computer players:

Time of use: Computer players drawing phase

Details:

Computer players will pick the ones with the most cards and draw his card in order to have the best chance to win the game. The ***max_index*** function determines which player the computer players should draw a card from, if there are multiple players with the most cards, it returns randomly among the players with the most cards.

Major Variables

Poker card representation: a tuple with a suit and a value e.g. (‘spades’, ‘K’)

self.game_finish:	a boolean showing the game ends or not
self.winners:	a list containing the winners of the game
self.player_list:	a 2-dimensional list containing the cards of all players, where the human player has the index 0
self.cards:	a list containing all cards of poker when the game initialized
loser:	an int value showing the index of the player that loses in self.player_list

Major Functions

partition(arr, fct, start, end):

partitions a part of an array from indices start to end with a function and returns a pivot index.

q_sort(arr, fct, start, end):

sorts a part of an array from indices start to end recursively with a function using the ***partition*** function.

my_quick_sort(arr, fct, start, end):

sorts an array with a function.

card_value(card_tuple):

takes a card tuple and returns an int value. If the argument is a number string, then returns converted value from str to int. If the argument is a string, then returns the sum of the ascii values of each character of the string.

max_index(l, index):

takes a 2-dimensional list and an index to skip, and returns a tuple of all indices of lists that have the most length excluding itself (the **index** argument).

game():

starts a game.

Major Methods

self.__init__(self, player_name, players):

initializes a game with arguments of the name of the human player and the total number of players in the game (human player included).

self.sort_cards(self):

sorts the cards of all players using ***my_quick_sort*** function.

self.throw_pairs(self):

examines the card lists of all players and remove cards with the same values determined by ***card_value*** function, and prints out messages according to the actions.

self.player_draw(self):

takes an input from the human player (console) and draw a card from the computer player determined by the input, then throws out the pairs in hand and prints out messages according to the actions.

self.player_turn(self):

examines whether the player has won the game or not first, 1. if he has 0 card in hand and has not been added in the self.winners list, then adds him into it and returns. 2. If he has already in the self.winners list, then returns. 3. If the losing condition is satisfied, then marks the boolean variable self.game_finish true and returns. After the examination, initializes the self.player_draw(self) method if all the 3 conditions above are not met.

Major Methods

self.com_turn(self):

examines the game ends or not, if so then returns. If not, then examines whether each of the computer players has won the game or not first, 1. if it has 0 card in hand and has not been added in the self.winners list, then adds it into it and returns. 2. If it has already in the self.winners list, then returns. 3. If the losing condition is satisfied, then marks the boolean variable self.game_finish true and returns. After the examination, initializes the drawing phase of the computer player using Drawing card algorithm of computer players mentioned in the algorithm section if all the 3 conditions above are not met, and then throws out the pairs in hand.

self.check_win(self, i):

checks whether a player with index i in self.player_list has won or not. 1. If it is already in self.winners list, returns true. 2. If it has 0 card in hand and has not yet been added to self.winners, adds it into self.winners and returns. 3. If it has cards in hand, returns false.

self.show_status(self):

prints out the cards that each player has.

self.show_final_result(self):

prints out the places of the winners and the final loser of the game.